Benchmarking out-of-memory strategies

Vincent Carey

Friday July 15 2016

Contents

1 Introduction

It is sometimes impossible to allocate large quantities of memory to do certain tasks in an R session:

- not enough RAM to represent a large object
  - can be a serious issue in HPC strategies with many CPUs with small endowment
- competing with other processes
  - can be a serious issue with multicore usage

Furthermore, it is often unnecessary to have random access to all elements of a large object

- requirements can be satisfied by scalable traversal (you choose chunk size, sequential vs. parallel, etc.)

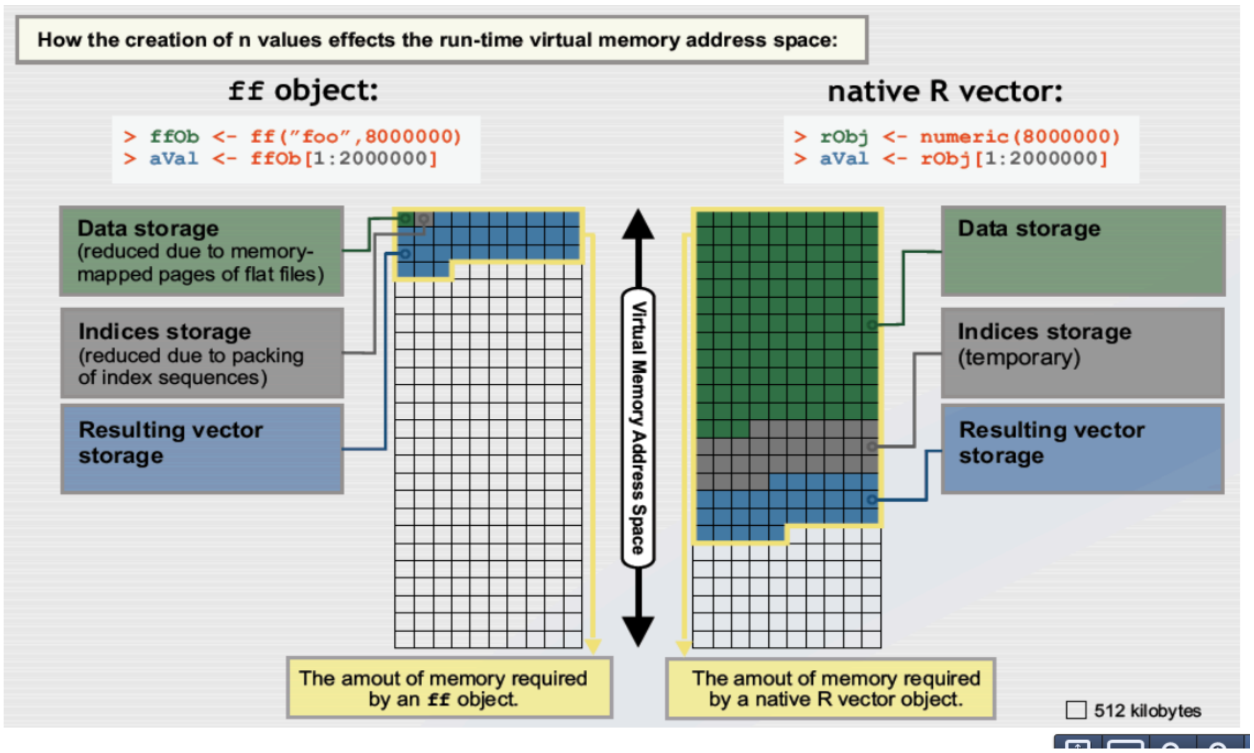
R has a reputation of being greedy with memory resources. Many packages have been devised to alleviate this concern:

- *ff*
- *bigmemory*

There has been some effort to adapt modeling algorithms to work naturally with these out-of-memory representations:

- *ffbase*
- *biglm*

Here's a schematic that describes the advantages of an approach like ff:


How the creation of n values effects the run-time virtual memory address space:
ff object:
> ffOb <- ff("foo",8000000)
> aVal <- ffOb[1:2000000]
native R vector:
> rObj <- numeric(8000000)
> aVal <- rObj[1:2000000]
Data storage (reduced due to memory-mapped pages of flat files)
Indices storage (reduced due to packing of index sequences)
Resulting vector storage
Virtual Memory Address Space
Data storage
Indices storage (temporary)
Resulting vector storage
The amout of memory required by an ff object.
The amout of memory required by a native R vector object.
512 kilobytes

We can also use general-purpose external stores for which custom interface packages exist:

- *rhdf5*
- *RSQLite*

Finally, there is often interest in the *data.table* infrastructure for handling large tables.

The purpose of this document is to discuss how to develop data on performance of these strategies to help decisionmaking:

- When should we adopt out-of-memory strategy?
- Which should we choose?
- How do we tune our procedures to perform best with the selected strategy?

2 A benchmarking function

The following function is a "proof of concept" that we can encapsulate the task of timing a variety of tasks (write to store, read contents of store, read a slice of 1000 records) for a variety of strategies (HDF5, ff, SQLite, data.table, bigmemory). The dot functions are hidden in this document and are very ad hoc. Several of the approaches are tunable through setting chunk sizes and these have been hard-coded to arbitrarily chosen values.

```
benchOOM
## function(NR=5000, NC=100, times=5, inseed=1234,
##   methods = list(.h5RoundTrip, .ffRoundTrip, .slRoundTrip, .dtRoundTrip, .bmRoundTrip)) {
## stopifnot(NR >= 5000) # some hardcoded indices... FIXME
## require(microbenchmark)
## nel = NR * NC
## set.seed(inseed)
## x = array(rnorm(nel), dim=c(NR,NC))
## cbind(NR=NR, NC=NC, unit="us", times=times, do.call(rbind,
##     lapply(methods, function(z) getStats(times, x, rtfun=z))))
## }
```

When we run the function we get some performance data. We can change aspects of the problem (number of rows, columns, number of repetitions) but need to attend to other aspects (such as effects of the environment) to generate really useful data on effectiveness of the approaches.

Should include native R storage as well. In this table 'wr' column tells how long it takes to write a matrix of given dimensions in the given format, 'ingFull' tells how long it takes to ingest the full matrix, 'ing1K' tells how long it takes to ingest a selection of 1000 contiguous rows (4001-5000).

```
benchOOM()
##     NR  NC unit times       meth         wr   ingFull     ing1K
## 1 5000 100   us     5       hdf5   8.787972  4.990620  5.897003
## 2 5000 100   us     5         ff 141.746789 16.725819  2.305410
## 3 5000 100   us     5     sqlite  68.016581 32.866247 13.386550
## 4 5000 100   us     5 data.table  26.298526  5.590558  4.103753
## 5 5000 100   us     5  bigmemory  31.280181  1.417235  0.282837
benchOOM(NR=100000)
##      NR  NC unit times       meth        wr   ingFull       ing1K
## 1 1e+05 100   us     5       hdf5  227.5885  33.61852   7.4404640
## 2 1e+05 100   us     5         ff 4280.6306 604.12610   2.6340938
## 3 1e+05 100   us     5     sqlite 1199.7374 839.63436 128.1789872
## 4 1e+05 100   us     5 data.table  385.4085 129.44778   3.9637442
## 5 1e+05 100   us     5  bigmemory  723.2483  29.27411   0.4027682
```

A disciplined approach to comparative evaluation would likely be a good candidate for the R Journal.